Practical 3\practical3.py

```python
import random
import threading
import time
from collections import deque

class ClientRequest:
    """Represents a client request with a unique ID and processing time."""
    def __init__(self, request_id):
        self.request_id = request_id
        self.arrival_time = time.time()
        self.processing_time = random.uniform(1, 3)  # Random processing time (1-3 sec)

class Server:
    """Represents a server handling requests."""
    def __init__(self, name, weight=1):
        self.name = name
        self.connections = 0  # Active connections
        self.weight = weight  # Used for Weighted Round Robin
        self.total_requests_handled = 0

    def handle_request(self, request):
        """Simulate handling a request."""
        self.connections += 1
        self.total_requests_handled += 1
        print(f" 🔹 Server {self.name} handling request {request.request_id} (Active: {self.connections})")
        time.sleep(request.processing_time)  # Simulate processing
        self.connections -= 1
        print(f"✅ Server {self.name} completed request {request.request_id} (Active: {self.connections})")

class LoadBalancer:
    """Implements multiple load balancing strategies."""
    def __init__(self, servers, strategy="round_robin"):
        self.servers = servers
        self.strategy = strategy
        self.lock = threading.Lock()
        self.request_queue = deque()
        self.total_requests = 0

        # Initialize Weighted Round Robin queue
        if strategy == "weighted_round_robin":
            self.weighted_queue = []
            for server in servers:
                self.weighted_queue.extend([server] * server.weight)

    def distribute_request(self, request):
        """Assign a request to a server based on the chosen strategy."""
        with self.lock:
            if self.strategy == "round_robin":
                server = self.request_queue.popleft()
                self.request_queue.append(server)

            elif self.strategy == "least_connections":
                server = min(self.servers, key=lambda s: s.connections)

            elif self.strategy == "weighted_round_robin":
                server = random.choice(self.weighted_queue)

            else:
                raise ValueError("Invalid load balancing strategy")

        # Process request in a new thread
        threading.Thread(target=server.handle_request, args=(request,)).start()
        self.total_requests += 1

def simulate_client_requests(load_balancer, num_requests):
    """Generates and distributes client requests."""
    for i in range(num_requests):
        request = ClientRequest(i+1)
        threading.Thread(target=load_balancer.distribute_request, args=(request,)).start()
        time.sleep(random.uniform(0.5, 1.5))  # Random inter-arrival time

# Create servers with different weights for Weighted Round Robin
server_list = [Server("S1", weight=2), Server("S2", weight=1), Server("S3", weight=3)]

# Choose a strategy: "round_robin", "least_connections", "weighted_round_robin"
strategy = "weighted_round_robin"  # Change as needed
load_balancer = LoadBalancer(server_list, strategy)

# Initialize Round Robin queue
if strategy == "round_robin":
    load_balancer.request_queue.extend(server_list)

# Run the simulation for 10 client requests
simulate_client_requests(load_balancer, 10)

# Wait for all requests to finish
time.sleep(10)

# Print statistics
print("\n📊 Load Balancing Summary:")
print(f"🔹 Total Requests Processed: {load_balancer.total_requests}")
for server in server_list:
    print(f"✅ {server.name}: {server.total_requests_handled} requests handled")

```

Output

```
Server S2 handling request 1 (Active: 1)
Server S3 handling request 2 (Active: 1)
Server S2 handling request 3 (Active: 2)
Server S2 completed request 1 (Active: 1)
Server S3 handling request 4 (Active: 2)
Server S2 completed request 3 (Active: 0)
Server S3 handling request 5 (Active: 3)
Server S3 completed request 2 (Active: 2)
Server S2 handling request 6 (Active: 1)
Server S3 completed request 4 (Active: 1)
Server S3 handling request 7 (Active: 2)
Server S3 handling request 8 (Active: 3)
Server S3 handling request 9 (Active: 4)
Server S3 completed request 5 (Active: 3)
Server S3 completed request 7 (Active: 2)
Server S2 completed request 6 (Active: 0)
Server S2 handling request 10 (Active: 1)
Server S3 completed request 9 (Active: 1)
Server S3 completed request 8 (Active: 0)
Server S2 completed request 10 (Active: 0)

Load Balancing Summary:
  Total Requests Processed: 10
  S1: 0 requests handled
  S2: 4 requests handled
  S3: 6 requests handled
```